ソフトウェア工学 124－3
（1999．10．21）

リアクティブプログラムの自動検証手法

山根 智†

リアクティブプログラムを形式的検証して，信頼性を保証することは，重要である．しかし，従来の研究では，状態遷移システムや抽象化されたプログラムの検証にとどまっており，プログラムの論理動作などの検証ができない．本論文では，リアクティブプログラムの操作的意味を状態遷移システムで形式化する立場より，リアクティブプログラムの自動検証手法を提案する．リアクティブプログラムのデータ領域を抽象化することにより，有限な状態遷移システムを構成して，自動検証を可能とする．最後に，実装した検証システムのプロトタイプにより，その有効性を示す．

Automatic Verification Method for Reactive Programs

SATOSHI YAMANE†

It is important to verify reactive programs and guarantee qualities. But in exisiting studies, there have been studies of verifications for only state transition systems and abstract programs. In this paper, we propose automatic verification method for reactive programs. Automatic verification is possible, because we construct finite state transiton systems from reactive programs by abstracting infinite data domains into finite domains. Finally, we show our methods effective by our prototype system for automatic verification.

1. まえがき

通信プロトコルや制御システムなどに組み込まれたリアクティブプログラムは，設計作業やプログラム動作が複雑であり，信頼性保証が困難である[1]．このために，リアクティブプログラムの仕様を状態遷移システムによりモデル化した形式的検証手法が盛んに研究されている．リアクティブプログラムの形式的検証のアプローチとしては，以下のようにフォワードアプローチとリバースアプローチに分類できる．

(1) フォワードアプローチ：上流工程の仕様を状態遷移システムとしてモデル化して，形式的検証するアプローチ
(2) リバースアプローチ：リアクティブプログラムから状態遷移システムを抽出して，形式的検証するアプローチ

従来の研究では，フォワードアプローチを基本としながら，以下のように，2つのアプローチから研究されている．

(1) フォワードアプローチ
  (a) Manna と Pnueli[2] は，簡易的な仕様記述言語 SPL(Simple Programming Language) を状態遷移システムでモデル化して，時相論理 PTL (Propositional Temporal Logic) の公理系を開発した．これにより，SPL が PTL 論理式を充足するかどうかを手作業で判定できる．しかし，SPL 言語がリアクティブプログラムとしては表現能力が低過ぎたり，検証が手作業であり，自動化できないでいない．
  (b) Clarke や Emerson ら[3] は，有限状態遷移システムが時相論理 CTL (Computation Tree Logic) 式を充足するかどうかを自動的に判定できるモデルチェッキングを開発した．しかし，有限状態遷移システムの検証に限定されており，リアクティブプログラムの検証ができない．
(2) リバースアプローチ
  (a) Corbett[4] は，Ada プログラムからタスクの同期とタイミング制約に関する情報を自動的に抽出して，ハイブリッドオートマトンとしてモデル化する手法を開発した．そして，オートマトン理論により，ハイブリッドオートマトンが安全性や活性を充足するかどうかを自動検証した．しかし，タスクの同期とタイミング制約に関する情報以外は隠蔽されており，プログラムの論理動作の検証ができない．さらに，大規模システムを検証するときに，状態空間組合せ爆発を発生させる．
  (b) Godefroid[5] は，C プログラムからタスクの同期に関する情報を自動的に抽出して，状態遷移システムとしてモデル化する手法を開発した．そして，状態空間組合せ爆発を抑制するために，半順序法 (Partial Order Method) による到達可能解析により，安全性の自動

† 〒890-0065 鹿児島市郡元1－21－40 鹿児島大学工学部情報工学科
Dept. of Information and Computer Science, Kagoshima University

検証を実現した．しかし，活性やプログラムの論理動作の検証ができない．

一般的に，産業界では，既存ソフトウェアを機能追加することが多いので，リバースアプローチが重要であると考えられる．本論文では，世の中に多く存在する既存プログラムの正当性が保証できるリバースアプローチの立場より，C プログラムから状態遷移システムを抽出して，時相論理で記述した安全性や活性の充足性を判定するモデルチェッキング検証手法を提案する．そのコンセプトを図1に示す．

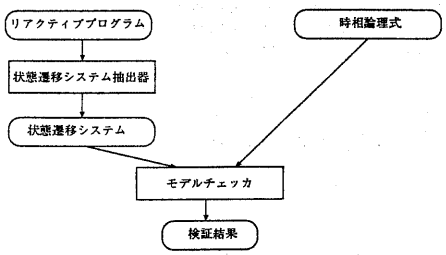

図1 リアクティブプログラムの形式的検証

一般的には，自然数や実数などの無限集合のデータを操作するプログラムは無限の状態空間を有する．本論文では，データ領域を有限集合に抽象化することにより，プログラムから有限状態遷移システムを構成して自動検証する手法を提案する．無限の状態空間を有する場合は，自動検証ができないために，演繹的証明手法により検証しなければならない．演繹的証明手法では，公理と推論規則から，会話的に定理を導出しながら証明する必要があり，大規模システムの検証が不可能である．

本論文では，リアクティブプログラムの状態集合を有限化するが，その動作は無限であるとする．なぜならば，リアクティブプログラムでは，外部の環境からの入力がランダムに連続的に発生して，停止性が仮定できないからである．

本論文では，まず，リアクティブCプログラムから状態遷移システムを抽出して，その操作的意味を状態遷移システム上で定義する．次に，プログラムを有限化するために，データ領域の抽象化手法を提案する．次に，安全性や活性を時相論理式で仕様記述して，状態遷移システムが時相論理式を充足するかどうかを，モデルチェッキング検証する手法を定義する．最後に，計算機実験により，本手法の有効性を示す．

本論文の構成は，次のとおりである．2章では，リアクティブCプログラムの操作的意味を状態遷移システムで定義する．3章では，状態空間を有限化する手法を提案して，モデルチェッキング検証手法を定義する．4章では，実現した検証システムのプロトタイプによる計算機実験を示す．最後に，5章では，まとめと今後の課題を述べる．

2. リアクティブCプログラムの操作的意味

リアクティブCプログラムの操作的意味は，状態遷移システム上で定義する．以下では，状態遷移システムを定義して，プログラムの操作的意味を状態遷移システムで表現する方法を定義する．さらに，リアクティブプログラムが充足すべき安全性や活性を表現する言語である時相論理を定義する．

2.1 状態遷移システム

まず，状態遷移システムを定義する．本論文で対象とする状態遷移システムは，一般化したKripke構造[6]である．

まず，変数の普遍集合 U とする．U の中の変数は，ブール値や整数などのような型付きである．状態 s は U の型整合解釈であり，各変数 $u \in U$ に値 $s[u]$ を割り付ける．ここで，すべての状態の集合を Σ とする．以下に，状態遷移システムを定義する．

定義 2.1 (状態遷移システム)
状態遷移システムは，$\mathbf{STS} = (V, \Theta, T)$ の3つ組で定義される．ここで，

(1) $V = \{u_1, \ldots, u_n\} \subseteq U$ はシステム変数の有限集合である．システム変数はデータ変数と状態変数から構成されており，データ変数はデータや制御データを表現して，状態変数はプログラムの状態，すなわち行番号を表現する．

(2) Θ は初期条件である．これは，すべての初期状態を特徴付ける表明である．状態が Θ を満足するならば，その状態は初期状態である．

(3) T は状態遷移の集合である．各状態遷移 $\tau \in T$ は関数 $\tau : \Sigma \rightarrow 2^{\Sigma}$ であり，各状態 $s \in \Sigma$ に次状態 τ-successor $\tau(s) \subseteq \Sigma$ を写像する．$\tau(s) \neq 0$ の時に限り状態遷移 τ は可能であり，そうでなければ τ は不可能である．状態遷移 τ に関連する関数は表明 $\rho_\tau(V, Vl)$ により表現される．$\rho_\tau(V, Vl)$ は状態遷移関係と呼び，状態 $s \in \Sigma$ を τ-successor $sl \in \tau(s)$ に関係付ける．なお，システム変数の値は s の中の値や sl の中の値として参照する．

■

$(s, sl) \models \rho_\tau(V, Vl)$ ならば，状態遷移関係 $\rho_\tau(V, Vl)$ は状態 sl を状態 s の τ-successor と見なす．ここで，(s, sl) は，$x \in V$ を $s[x]$ として解釈して，xl を $sl[x]$ として解釈する．

状態遷移 τ の可能性は，$En(\tau) : \exists Vl.\rho_\tau(V, Vl)$ により，表現される．$En(\tau)$ は，s がある τ-successor を持つ時に限り，真である．

次に，状態遷移システムの計算を定義する．

定義 2.2 (状態遷移システムの計算)
状態遷移システム $\mathbf{STS} = (V, \Theta, T)$ に対して，以下の条件を満たすならば，状態の無限列 $\sigma = s_0, s_1, s_2, \ldots$ は $\mathbf{STS}$ の計算である．

(1) 初期条件
s_0 が初期状態である．すなわち，$s_0 \models \Theta$．

(2) 連続条件
任意の $j = 0, 1, \ldots$ に対して，状態 s_{j+1} は状態 s_j の

τ-successor である．すなわち，ある $\tau \in T$ に対して，$s_{j+1} \in \tau(s_j)$ である．

■

通信プロトコルやオペレーティングシステムなどのリアクティブシステムは停止性が仮定できないので，状態遷移システムの状態列が初期条件と連続条件を満たすことを要求するのは妥当である．

2.2 リアクティブプログラムの状態遷移システム表現

まず，図2のような簡単なエアコン制御システムを考える．その仕様は以下のとおりである．エアコン制御プログラムは，センサ感知プログラムより送信される，温度計とドアに関するメッセージに反応する．すなわち，部屋がhotかcoolのときか，ドアがclosedかopenedのときに，メッセージが送信される．部屋がhotでドアがclosedのときには，アクチュエータ駆動プログラムにメッセージを送信してエアコンを作動して，そうでないときはエアコンを停止するようにプログラムは動作する．

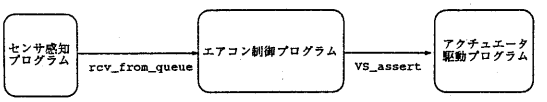

図2 エアコン制御システム

以下に，図2のエアコン制御システムを制御するプログラムを示す．

```
 1 void AC_controller()
 2 {
 3 char *message=null;
 4    /* initially, */
 5 int is_room_hot=0;     /* room is not hot */
 6 int is_door_closed=1; /* door is closed */
 7 int ac=0;              /* so, ac is off */
 8 while (1) {
 9   message=(char *)rcv_from_queue(to_me,QSZ);
10   if (strcmp(message,"room_is_hot") == 0) {
11       is_room_hot=1;
12   };
13   if (strcmp(message,"room_is_cool") == 0) {
14       is_room_hot=0;
15   };
16   if (strcmp(message,"open_door") == 0) {
17       is_door_closed=0;
18       ac=0;
19   };
20   if ((strcmp(message,"close_door") == 0)){
21       is_door_closed=1;
22       if (is_room_hot)
23           ac=1;
24   };
25   /* test */
26   if (is_room_hot && is_door_closed){
27       VS_assert(ac);
28   };
29 }
```

図3 エアコン制御プログラム

このプログラムを状態遷移システム $\mathbf{STS} = (V, \Theta, T)$ として表現する．ただし，to_me と QSZ はプログラム動作に影響を与えないので，以下では省略する．7行目まではデータ定義なので，8行目を実行開始点とする．

(1) $V = \{u_1, message, is\_room\_hot, is\_door\_closed, ac\}$．ただし，$u_1$ は行番号を表す．

(2) $\Theta = (u_1 = 8 \wedge message = null \wedge is\_room\_hot = 0 \wedge is\_door\_closed = 1 \wedge ac = 0)$

(3) $\tau \in T$ を定義する．

(a) $\tau((u_1 = 8 \wedge message = null \wedge is\_room\_hot = 0 \wedge is\_door\_closed = 1 \wedge ac = 0)) = \{(u_1 = 9 \wedge message = room\_is\_hot \wedge is\_room\_hot = 0 \wedge is\_door\_closed = 1 \wedge ac = 0)\}$

(b) $\tau((u_1 = 9 \wedge message = room\_is\_hot \wedge is\_room\_hot = 0 \wedge is\_door\_closed = 1 \wedge ac = 0)) = \{(u_1 = 10 \wedge message = room\_is\_hot \wedge is\_room\_hot = 0 \wedge is\_door\_closed = 1 \wedge ac = 0)\}$

(c) $\tau((u_1 = 10 \wedge message = room\_is\_hot \wedge is\_room\_hot = 0 \wedge is\_door\_closed = 1 \wedge ac = 0)) = \{(u_1 = 11 \wedge message = room\_is\_hot \wedge is\_room\_hot = 1 \wedge is\_door\_closed = 1 \wedge ac = 0)\}$

.....

2.3 時相論理

本論文では，時相論理 $ACTL$ により，リアクティブプログラムの充足すべき安全性や活性を表現する．重要な性質は時相論理 $ACTL$ により，すべて表現できることが知られている．$ACTL$ は，分岐時相論理 CTL[6] において，パス限量記号 A のみを許す時相論理である．以下に，時相論理 $ACTL$ の構文と意味を定義する．

定義 2.3 ($ACTL$ の構文)

$ACTL$ の論理式の集合は，次の規則で生成される論理式の集合のうち最小のものである．

(1) すべての原子命題 $p, \neg p \in AP$ は $ACTL$ 論理式である．ここで，AP は原子命題の集合である．

(2) f_1 と f_2 が $ACTL$ 論理式ならば，$f_1 \wedge f_2$, $f_1 \vee f_2$, AXf_1, $A[f_1 U f_2]$, は $ACTL$ 論理式である．ただし，A(あらゆる未来で) はパス限量記号，X は NEXT オペレータ，U は until オペレータである．

■

$ACTL$論理式fのモデルは, 状態遷移システム $\mathbf{STS}=(V,\Theta,T)$ の無限の状態列 σ=$s_0,s_1,s_2,\ldots$ である. 以下に, モデルに対する論理式の意味を定義する.

定義 2.4 ($ACTL$の意味)

モデルσに対して, $ACTL$論理式fがモデルの位置$j\geq 0$で成り立つことを$\sigma,j\models f$と表現して, 以下に帰納的に定義する.

(1) $\sigma,j\models p$ iff $p\in L(s_j)$.
$\sigma,j\models\neg p$ iff $p\notin L(s_j)$.

(2) $\sigma,j\models f_1\wedge f_2$ iff $\sigma,j\models f_1$ かつ $\sigma,j\models f_2$.
$\sigma,j\models f_1\vee f_2$ iff $\sigma,j\models f_1$ または $\sigma,j\models f_2$.

(3) $\sigma,j\models AXf_1$ iff すべての$j+1$に対して, $\sigma,j+1\models f_1$.

(4) $\sigma,j\models A[f_1Uf_2]$ iff すべての$s_j,s_{j+1},\ldots,s_i$に対して, $\exists i[i\geq 0\wedge\sigma,i\models f_2\wedge\forall j[0\leq j<i\rightarrow\sigma,j\models f_1]]$.

ただし, $AFf\equiv A[trueUf]$とする. ■

3. 状態遷移システムの有限化と自動検証

リアクティブプログラムから構成される状態遷移システムは無限な状態遷移システムであり, 自動検証手法が適用できない. そこで, 抽象実行 (Abstract Interpretation) の中で使われたデータ構造の抽象化[7] により, データ領域を有限化して, 有限な状態遷移システムを構成する手法を提案する.

3.1 無限状態遷移システムの抽象化

抽象実行 (Abstract Interpretation) の中で使われたデータ構造の抽象化[7] により, データ領域を有限化して, 有限な状態遷移システムを構成する手法を提案する.

まず, 無限状態遷移システムから, 有限状態遷移システムの構成方法を定義する.

定義 3.1 (有限状態遷移システムの構成)

無限状態遷移システム $\mathbf{STS^C}=(V^C,\Theta^C,T^C)$ から, 有限状態遷移システム $\mathbf{STS^A}=(V^A,\Theta^A,T^A)$ を構成する. その構成方法は, 以下のとおりである.

(1) ${v_1}^C,{v_2}^C,\ldots,{v_n}^C\in V^C$ をブール変数${v_1}^A,{v_2}^A,\ldots,{v_m}^A\in V^A$ に置換する.

(2) Θ^C の中の変数V^Cを変数V^Aに置換してΘ^Aとする.

(3) T^C の中の変数V^Cを変数V^Aに置換してT^Aとする.

■

次に, 以降の議論を容易にするために, 状態遷移システムをKripke構造として定義しなおす.

定義 3.2 (Kripke構造)

状態遷移システム $\mathbf{STS}=(V,\Theta,T)$ を Kripke 構造 $\mathbf{K}=(\Sigma,\Sigma_0,R,L)$ として定義しなおす.

(1) Σは状態の有限集合であり, $s\in\Sigma$はVの型整合解釈である.

(2) Σ_0は初期状態集合であり, ${s_0}\in\Sigma_0$は$s_0\models\Theta$を満たす状態である.

(3) $R\subseteq\Sigma\times\Sigma$は状態遷移関係であり, $\tau\in T$に対して $\tau(s_1)=\{s_2\}$ ならば $R(s_1,s_2)$ である.

(4) $L:\Sigma\rightarrow 2^{AP}$ は状態に原子命題の集合をラベル付けする関数である. ただし, 原子命題の集合 AP は, V の型整合解釈の部分集合である.

■

ここで, 前述の状態遷移システム $\mathbf{STS}=(V,\Theta,T)$ を Kripke 構造 $\mathbf{K}=(\Sigma,\Sigma_0,R,L)$ として定義しなおす. この Kripke 構造 $\mathbf{K}$ は, 以降で $\mathbf{K^C}$ と参照するので, $\mathbf{K^C}=({\Sigma^C},{\Sigma_0}^C,R^C,L^C)$ とする.

(1) $\Sigma^C=\{(u_1=8\wedge message=null\wedge is\_room\_hot=0\wedge is\_door\_closed=1\wedge ac=0),(u_1=9\wedge message=room\_is\_hot\wedge is\_room\_hot=0\wedge is\_door\_closed=1\wedge ac=0),\ldots\}$

(2) ${s_0}^C=(u_1=8\wedge message=null\wedge is\_room\_hot=0\wedge is\_door\_closed=1\wedge ac=0)$, ただし ${s_0}^C\in{\Sigma_0}^C$ である.

(3) R^C を定義する.

(a) $R^C((u_1=8\wedge message=null\wedge is\_room\_hot=0\wedge is\_door\_closed=1\wedge ac=0),(u_1=9\wedge message=room\_is\_hot\wedge is\_room\_hot=0\wedge is\_door\_closed=1\wedge ac=0))$

(b) $R^C((u_1=9\wedge message=room\_is\_hot\wedge is\_room\_hot=0\wedge is\_door\_closed=1\wedge ac=0),(u_1=10\wedge message=room\_is\_hot\wedge is\_room\_hot=0\wedge is\_door\_closed=1\wedge ac=0))$

(c) $R^C((u_1=10\wedge message=room\_is\_hot\wedge is\_room\_hot=0\wedge is\_door\_closed=1\wedge ac=0),(u_1=11\wedge message=room\_is\_hot\wedge is\_room\_hot=1\wedge is\_door\_closed=1\wedge ac=0))$

.....

(4) 例えば, L^C は以下のように定義できる.

(a) 例えば, 行番号 8 において L^C は以下である. $L^C(message,is\_room\_hot,is\_door\_closed,ac)=(message=null\wedge is\_room\_hot=0\wedge is\_door\_closed=1\wedge ac=0)$

(b) 例えば, 行番号 9 において L^C は以下である. $L^C(message,is\_room\_hot,is\_door\_closed,ac)=(message=room\_is\_hot\wedge is\_room\_hot=0\wedge is\_door\_closed=1\wedge ac=0)$

.....

次に, 抽象化した有限状態の Kripke 構造が満たすべき望ましい性質を定義する.

定理 3.1 (抽象化による $ACTL$ の保存)

無限状態の Kripke 構造 $\mathbf{K^C}=({\Sigma^C},{\Sigma_0}^C,R^C,L^C)$ から, 有限状

態のKripke構造 $\mathbf{K^A} = (\Sigma^A, {\Sigma_0}^A, R^A, L^A)$ が構成できたとする．ここで，具体化関数 $\gamma : \Sigma^A \to 2^{\Sigma^C}$ が以下の条件を満たすとする．

(1) 初期条件：${\Sigma_0}^C \subseteq \gamma({\Sigma_0}^A)$

(2) 連続条件：ある ${s_1}^C \in \gamma({s_1}^A)$ と ${s_2}^C \in \gamma({s_2}^A)$ に対して，$R^C({s_1}^C, {s_2}^C)$ ならば，$R^A({s_1}^A, {s_2}^A)$ である．

このとき，$ACTL$ 論理式 f に対して，

$\sigma^A, j \models f$ ならば $\sigma^C, j \models \gamma(f)$

が成り立つ．

証明3.1 $ACTL$ 論理式の構造に関する帰納法で証明する．

(1) $f = p \in AP$ のとき
もし，$\sigma^A, j \models p$ ならば，$p \in L^A({s_j}^A)$ であるような状態 ${s_j}^A$ が存在する．このとき，関数 γ により，状態 ${s_j}^C \in \gamma({s_j}^A)$ が存在する．また，関数 γ は V^A から V^C への写像であり，状態 ${s_j}^A$ 上で充足される p も関数 γ により具体化されて，$\gamma(p) \in L^C({s_j}^C)$ となる．以上より，$\sigma^C, j \models \gamma(p)$ が成り立つ．

(2) $f = \neg p \in AP$ のとき
$f = p$ のときと同様に証明できる．

(3) $f = f_1 \wedge f_2$ のとき
もし，$\sigma^A, j \models f_1 \wedge f_2$ ならば，$\sigma^A, j \models f_1$ かつ $\sigma^A, j \models f_2$ である．帰納法の仮定より，$\sigma^C, j \models \gamma(f_1)$ かつ $\sigma^C, j \models \gamma(f_2)$ である．ゆえに，$\sigma^C, j \models \gamma(f_1 \wedge f_2)$ である．

(4) $f = f_1 \vee f_2$ のとき
$f = f_1 \wedge f_2$ のときと同様に証明できる．

(5) $f = AXf_1$ のとき
もし，$\sigma^A, j \models AXf_1$ ならば ${s_j}^A \models AXf_1$ であり，$R^A({s_j}^A, {s_{j+1}}^A)$ を満たす任意の ${s_{j+1}}^A$ に対して，${s_{j+1}}^A \models f_1$ である．帰納法の仮定より，${s_{j+1}}^C \models \gamma(f_1)$ である．これより，${s_j}^C \models AX\gamma(f_1)$ であり，${s_j}^C \models \gamma(AXf_1)$ である．

(6) $f = A[f_1 U f_2]$ のとき
もし，$\sigma^A, j \models A[f_1 U f_2]$ ならば，すべての ${s_j}^A, {s_{j+1}}^A, \ldots, {s_i}^A$ に対して，$\exists i[i \geq 0 \wedge \sigma^A, i \models f_2 \wedge \forall j[0 \leq j < i \to \sigma^A, j \models f_1]]$ である．帰納法の仮定より，$\sigma^C, j \models \gamma(A[f_1 U f_2])$ である．

■

前述のエアコン制御プログラムを例に説明する．まず，以下のように変数 is_room_hot と is_door_closed, ac, $message$ を抽象化する．

(1) is_room_hot はブール変数 $b^{is\_room\_hot}$ と抽象化する．$is\_room\_hot = 0$ であれば $b^{is\_room\_hot} = 0$, $is\_room\_hot \neq 0$ であれば $b^{is\_room\_hot} = 1$ とする．

(2) is_door_closed はブール変数 $b^{is\_door\_closed}$ と抽象化する．$is\_door\_closed = 0$ であれば $b^{is\_door\_closed} = 0$, $is\_door\_closed \neq 0$ であれば $b^{is\_door\_closed} = 1$ とする．

(3) ac はブール変数 b^{ac} と抽象化する．$ac = 0$ であれば $b^{ac} = 0$, $ac \neq 0$ であれば $b^{ac} = 1$ とする．

(4) $message$ はブール変数 $b^{message=room\_is\_hot}$, $b^{message=room\_is\_cool}$, $b^{message=open\_door}$, $b^{message=close\_door}$ とする．$message = room\_is\_hot$ のときのみ $b^{message=room\_is\_hot} = 1$, $message = room\_is\_cool$ のときのみ $b^{message=room\_is\_cool} = 1$, $message = open\_door$ のときのみ $b^{message=open\_door} = 1$, $message = close\_door$ のときのみ $b^{message=close\_door} = 1$ である．

本来のC言語にはブール型はないが，ここではブール型を導入してC言語風に抽象化したプログラムを記述する．

```
 1 void AC_controller()
 2 {
 3 bool b^{message=room_is_hot}=0,
        b^{message=room_is_cool}=0,
        b^{message=open_door}=0,
        b^{message=close_door}=0;
 4    /* initially, */
 5 bool b^{is_room_hot}=0;      /* room is not hot */
 6 bool b^{is_door_closed}=1;   /* door is closed */
 7 bool b^{ac}=0;               /* so, ac is off */
 8 while (1) {
 9   b^{message=room_is_hot},b^{message=room_is_cool},
     b^{message=open_door},b^{message=close_door}
     のどれか一つを 1 にする．
10   if (b^{message=room_is_hot} == 1) {
11       b^{is_room_hot}=1;
12   };
13   if (b^{message=room_is_cool} == 1) {
14       b^{is_room_hot}=0;
15   };
16   if (b^{message=open_door} == 1) {
17       b^{is_door_closed}=0;
18       b^{ac}=0;
19   };
20   if (b^{message=close_door} == 1){
21       b^{is_door_closed}=1;
22       if (b^{is_room_hot})
23           b^{ac}=1;
24   };
25   /* test */
26   if (b^{is_room_hot} && b^{is_door_closed}){
27       VS_assert(ac);
28   };
29 }
```

図4 抽象化したエアコン制御プログラム

このプログラムを状態遷移システム $\mathbf{STS^A} = (V^A, \Theta^A, T^A)$ として表現する.

(1) $V^A = \{u_1, b^{message=room\_is\_hot}, b^{message=room\_is\_cool}, b^{message=open\_door}, b^{message=close\_door}, b^{is\_room\_hot}, b^{is\_door\_closed}, b^{ac}\}$. ただし, u_1 は行番号を表す.

(2) $\Theta^A = (u_1 = 8 \wedge b^{message=room\_is\_hot} = 0 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0)$

(3) $\tau^A \in T^A$ を定義する.

(a) $\tau^A((u_1 = 8 \wedge b^{message=room\_is\_hot} = 0 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0)) = \{(u_1 = 9 \wedge b^{message=room\_is\_hot} = 1 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0)\}$

(b) $\tau^A((u_1 = 9 \wedge b^{message=room\_is\_hot} = 1 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0)) = \{(u_1 = 10 \wedge b^{message=room\_is\_hot} = 1 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0)\}$

(c) $\tau^A((u_1 = 10 \wedge b^{message=room\_is\_hot} = 1 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0)) = \{(u_1 = 11 \wedge b^{message=room\_is\_hot} = 1 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 1 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0)\}$

.....

さらに, これを Kripke 構造 $\mathbf{K^A} = (\Sigma^A, {\Sigma_0}^A, R^A, L^A)$ として定義しなおす.

(1) $\Sigma^A = \{(u_1 = 8 \wedge b^{message=room\_is\_hot} = 0 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0), (u_1 = 9 \wedge b^{message=room\_is\_hot} = 1 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0), \ldots\}$

(2) ${s_0}^A = (u_1 = 8 \wedge b^{message=room\_is\_hot} = 0 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0)$, ただし ${s_0}^A \in {\Sigma_0}^A$ である.

(3) R^A を定義する.

(a) $R^A((u_1 = 8 \wedge b^{message=room\_is\_hot} = 0 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0), (u_1 = 9 \wedge b^{message=room\_is\_hot} = 1 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0))$

(b) $R^A((u_1 = 9 \wedge b^{message=room\_is\_hot} = 1 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0), (u_1 = 10 \wedge b^{message=room\_is\_hot} = 1 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0))$

(c) $R^A((u_1 = 10 \wedge b^{message=room\_is\_hot} = 1 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0), (u_1 = 11 \wedge b^{message=room\_is\_hot} = 1 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 1 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0))$

.....

(4) 例えば, L^A は以下のように定義できる.

(a) 例えば, 行番号 8 において L^A は以下である.
$L^A(b^{message=room\_is\_hot}, b^{message=room\_is\_cool}, b^{message=open\_door}, b^{message=close\_door}, b^{is\_room\_hot}, b^{is\_door\_closed}, b^{ac}) = (b^{message=room\_is\_hot} = 0 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0)$

(b) 例えば, 行番号 9 において L^A は以下である.
$L^A(b^{message=room\_is\_hot}, b^{message=room\_is\_cool}, b^{message=open\_door}, b^{message=close\_door}, b^{is\_room\_hot}, b^{is\_door\_closed}, b^{ac}) = (b^{message=room\_is\_hot} = 1 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0)$

.....

次に, 無限状態の Kripke 構造 $\mathbf{K^C} = (\Sigma^C, {\Sigma_0}^C, R^C, L^C)$ から構成した, 有限状態の Kripke 構造 $\mathbf{K^A} = (\Sigma^A, {\Sigma_0}^A, R^A, L^A)$ に対して, 具体化関数 $\gamma : \Sigma^A \to 2^{\Sigma^C}$ が以下の条件を満たすことを確認する.

(1) 初期条件 : ${\Sigma_0}^C \subseteq \gamma({\Sigma_0}^A)$

(2) 連続条件 : ある ${s_1}^C \in \gamma({s_1}^A)$ と ${s_2}^C \in \gamma({s_2}^A)$ に対して, $R^C({s_1}^C, {s_2}^C)$ ならば, $R^A({s_1}^A, {s_2}^A)$ である.

まず, $\gamma : \Sigma^A \to 2^{\Sigma^C}$ を定義する.

(1) $\gamma((u_1 = 8 \wedge b^{message=room\_is\_hot} = 0 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0)) = \{(u_1 = 8 \wedge message = \perp \wedge is\_room\_hot = 0 \wedge is\_door\_closed = 1 \wedge ac = 0)\}$

(2) $\gamma((u_1 = 9 \wedge b^{message=room\_is\_hot} = 1 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0)) = \{(u_1 = 9 \wedge message = room\_is\_hot \wedge is\_room\_hot = 0 \wedge is\_door\_closed = 1 \wedge ac = 0)\}$

(3) $\gamma((u_1 = 10 \wedge b^{message=room\_is\_hot} = 1 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 0 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0)) = \{(u_1 = 10 \wedge message = room\_is\_hot \wedge is\_room\_hot = 0 \wedge is\_door\_closed = 1 \wedge ac = 0)\}$

(4) $\gamma((u_1 = 11 \wedge b^{message=room\_is\_hot} = 1 \wedge b^{message=room\_is\_cool} = 0 \wedge b^{message=open\_door} = 0 \wedge b^{message=close\_door} = 0 \wedge b^{is\_room\_hot} = 1 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0)) = \{(u_1 = 11 \wedge message = room\_is\_hot \wedge is\_room\_hot = 1 \wedge is\_door\_closed = 1 \wedge ac = 0)\}$

.....

このように, $\mathbf{K^A}$ と $\mathbf{K^C}$ の状態は 1 対 1 対応しており, さらに状態遷移関係も 1 対 1 対応している. 以上より, 初期条件と連続条件を満たすことは明白である. これより, $ACTL$ 論理式 f に対して,

$\sigma^A, j \models f$ ならば $\sigma^C, j \models \gamma(f)$

が成り立つことがわかる.

3.2 有限状態遷移システムのモデルチェッキング検証

有限化した状態遷移システムをモデルチェッキング検証することにより, 元のプログラムの正当性を検証する. 以下に, 有限化したリアクティブ C プログラムのモデルチェッキング検証の事例を示す.

まず, 検証すべき安全性と活性を示す.

(1) 安全性

安全性としては, 常に, 部屋の温度が暑くて部屋が開いているときはエアコンが作動しないといった性質である. 時相論理式で仕様記述すると, 以下のようになる.

$AG\neg p_{hotopen}$

ここで, $p_{hotopen} = (b^{is\_room\_hot} = 1 \wedge b^{is\_door\_closed} = 0 \wedge b^{ac} = 1)$ である.

(2) 活性

活性としては, いつかは部屋は涼しくなるといった性質である. 時相論理式で仕様記述するすると, 以下のようになる.

$AG(p_{hot} \to AF(q_{aircon} \vee r_{open}))$

ここで, $p_{hot} = (b^{is\_room\_hot} = 1 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 0)$, $q_{aircon} = (b^{is\_room\_hot} = 1 \wedge b^{is\_door\_closed} = 1 \wedge b^{ac} = 1)$, $r_{open} = (b^{is\_room\_hot} = 1 \wedge b^{is\_door\_closed} = 0 \wedge b^{ac} = 0)$ である.

以下に, 活性 $AG(p_{hot} \to AF(q_{aircon} \vee r_{open}))$ のモデルチェッキング検証の例を示す. $AG(p_{hot} \to AF(q_{aircon} \vee r_{open}))$ は $AG(\neg p_{hot} \vee AF(q_{aircon} \vee r_{open}))$ と考えることができるので, $ACTL$ 時相論理式である. 原子命題 $\neg p_{hot}$, q_{aircon}, r_{open} がラベル付いた状態遷移システムに対して, 以下の手順で部分論理式が成立する状態集合をラベリングする.

(1) まず, $q_{aircon} \vee r_{open}$ が成立する状態集合をラベリングする.

(2) 次に, $AF(q_{aircon} \vee r_{open})$ が成立する状態集合をラベリングする.

(3) 次に, $\neg p_{hot} \vee AF(q_{aircon} \vee r_{open})$ が成立する状態集合をラベリングする.

(4) 最後に, $AG(\neg p_{hot} \vee AF(q_{aircon} \vee r_{open}))$ が成立する状態集合をラベリングする.

そして, 初期状態が $AG(\neg p_{hot} \vee AF(q_{aircon} \vee r_{open}))$ でラベリングされていれば, 時相論理式 $AG(p_{hot} \to AF(q_{aircon} \vee r_{open}))$ が成立すると考える.

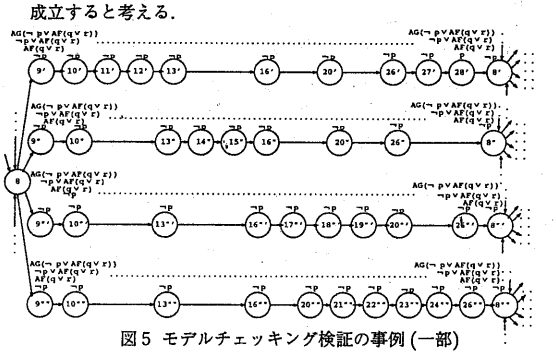

図 5 モデルチェッキング検証の事例 (一部)

以上のモデルチェッキングアルゴリズムの様子の一部を図 5 に示す. 図 5 より, $AG(p_{hot} \to AF(q_{aircon} \vee r_{open}))$ は充足する. ゆえに, 元の無限状態遷移システムにおいて, $\gamma(AG(p_{hot} \to AF(q_{aircon} \vee r_{open})))$, すなわち, $AG({p_{hot}}^C \to AF({q_{aircon}}^C \vee {r_{open}}^C))$ が充足される. ここで,

${p_{hot}}^C = (is\_room\_hot = 1 \wedge is\_door\_closed = 1 \wedge ac = 0)$,
${q_{aircon}}^C = (is\_room\_hot = 1 \wedge is\_door\_closed = 1 \wedge ac = 1)$,
${r_{open}}^C = (is\_room\_hot = 1 \wedge is\_door\_closed = 0 \wedge ac = 0)$
である.

4. リアクティブプログラムの形式的検証システム

提案した手法を支援する検証システムのプロトタイプを実現した. 検証システムは, SUN4/IP(メモリ 8MB, ディスク 160MB) 上で動作して, C 言語で約 3000 ステップである.

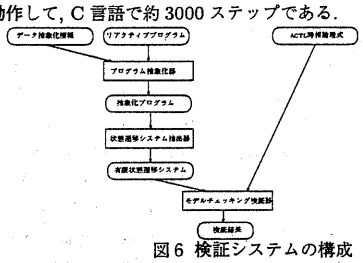

図 6 検証システムの構成

検証システムは, 図 6 に示すように, プログラム抽象化器と状態遷移システム抽出器, モデルチェッキング検証器から構成される. プログラム抽象化器は, データ抽象化情報を入力して, リアクティブプログラムから抽象化されたプログラムを構成するものである. 状態遷移システム抽出器は, 抽象化されたプログラムから有限状態遷移システムを構成するものである. 今回実現したプログラム抽象化器と状態遷移システム抽出器はプロトタイプである. プログラム抽象化器は, 前章の例のように, 元のプログラム実行文と抽象化したプログラムの実行文が 1 対 1 対応した場合にのみ適用可能である. 状態遷移システム抽出器は, プログラムをインタプリトしながら, 有限状態遷移システムを構成するものである.

検証コストを評価するために, 前章におけるプロセスが並行に動作する場合に計算機実験を行った. 検証した安全性や活性は前章と同様であり, $AG\neg({p_{hotopen}}^1 \wedge {p_{hotopen}}^2 \wedge \ldots \wedge {p_{hotopen}}^n)$ や $AG({p_{hot}}^1 \rightarrow AF({q_{aircon}}^1 \vee {r_{open}}^1)) \wedge AG({p_{hot}}^2 \rightarrow AF({q_{aircon}}^2 \vee {r_{open}}^2)) \wedge \ldots \wedge AG({p_{hot}}^b \rightarrow AF({q_{aircon}}^n \vee {r_{open}}^n))$ である. ここで, ${p_{hotopen}}^i$ や ${p_{hot}}^i$, ${q_{aircon}}^i$, ${r_{open}}^i$ はプロセス i の原子命題である. 安全性と活性の検証コストはほぼ同じであり, プロセス数に対して図 7 のようになった. プロセス数に対して, 指数的に状態数と遷移数が増加するので, 計算時間と所要メモリ量も指数的に増加した.

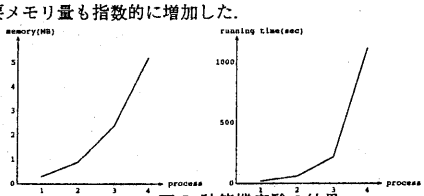

図 7 計算機実験の結果

5. むすび

本論文では, リアクティブプログラムの操作的意味を状態遷移システムで表現する方法に着目して, リアクティブプログラムを抽象化することによって, 有限状態遷移システムを構成して自動検証する手法を提案した. 類似研究として, E.M. Clarke らは状態遷移システムを抽象化してモデルチェッキング検証する手法を研究している[8]. E.M. Clarke らはハードウェアの構造に関する準同型写像を定義して, 抽象化された状態遷移システムを構成する方法を提案している. 本論文はプログラム変数をブール変数で抽象化する手法であり, プログラムの抽象化に適している. また, C. Loiseaux らは, 具体的な状態遷移システムと抽象化した状態遷移システムがどのような条件により, $\mu - calculus$ を保存するかを理論的に研究した[9]. しかし, $\mu - calculus$ の保存関係による抽象化では, 抽象化される状態遷移システムが大きすぎる. さらに, C. Loiseaux らは, 抽象化状態遷移システムを構成する方法を与えていない. また, D. Dams らは, 簡易なガード型言語を対象として, $\mu - calculus$ を保存する抽象的な状態遷移システムの性質や構成方法を研究している[10]. しかし, 対象とする言語が簡易であったり, 構成方法がアドホックであり, 一般的でない. 以上より, 本論文のように, プログラムを対象とした抽象化した有限状態遷移システムを構成して, 自動検証する研究は存在しない. 今後の研究課題としては, 実用的なプログラムを対象とした計算機実験及びモジュール毎に検証する構成的手法 (Compositional Verification) を考慮した並列プログラムの効率的な自動検証などが考えられる.

参考文献

1) D. Harel, A. Pnueli. *On the development of reactive systems.* NATO ASI Series, pp.477-498, 1985.
2) Z. Manna, A. Pnueli. *Temporal Verification of Reactive Systems.* Springer-Verlag, P. 512, 1995.
3) E.M. Clarke, et-al.. Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications. *ACM TOPLAS*, No.8, Vol.12, pp. 244-263, 1986.
4) J.C. Corbett. Timing Analysis of Ada Tasking Programs. *IEEE Trans. SE*, No.22, Vol.7, pp. 461-483, 1996.
5) P. Godefroid. Model Checking for Programming Languages using VeriSoft. *ACM POPL*, pp. 174-186, 1997.
6) E.A. Emerson. *Temporal and Modal Logic.* Handbook of TCS, Vol.B, pp. 995-1072, 1990.
7) P. Cousot, R. Cousot. Abstarct interpretation. *ACM POPL*, pp. 238-252, 1977
8) E.M. Clarke, et-al. Model Checking and Abstraction. *ACM TOPLAS*, No.16, Vol.5, pp. 1512-1542, 1994.
9) C. Loiseaux, et-al. Property Preseving Abstractions for the Verification of Concurrent Systems. *Formal Methods in System Design*, No.6, pp.11-44, 1995.
10) D. Dams, et-al. Abstract Interpretation of Reactive Systems. *ACM TOPLAS*, No.19, Vol.2, pp. 253-291, 1997.